SQL Database: Golf Statistics

Alan Lee

05/01/2013

Table of Contents

Project Direction Overview 3

Use Cases and Fields 4

Structural Database Rules 7

Entity – Relationship Diagram 9

Specialization – Generalization Relationships 9

Initial DBMS Physical ERD 12

Physical ERD 12

Updated Physical ERD 23

Normalization 24

Entity Check 25

Tables Created 26

Index Placement 27

Creating Indexes in SQL 32

Procedures 34

Updating Handicap Level Through Trigger 36

Creating Questions and Queries for the Database 39

Summary and Reflection 44

Project Direction Overview

I have played golf since I was three years old and played golf professionally when I graduated from college. Needless to say, I have been surrounded by a plethora of golfers and many of them had no structured practice like in other team sports. I would like to develop an app and a website that allows golfers to access their data and view amateur golfing data around the world. This will give the players a sense of how they could practice more effectively instead of just hitting balls onto the driving range.   They can measure their improvements by a handicap level or by looking at their scoring average for a specific golf course (golf courses vary in difficulty so scores will fluctuate as well). This golfing database will show the golfer how they are compared to other people that are better than them. If they choose to emulate their statistics in a certain area, then they can focus on the specific area when they practice.

There are many ways for a golfer to improve their game. For example, they can record their data or scores into the database, load them into the app/website and see how they can compare with other golfers with the same handicap bracket. For instance, if a golfer is a handicap 10, then they would fall into the category of 9-13 handicap. After playing 18 holes, the player would input their statistics; they can review their data in the website and compare themselves with other people. From there on, the golfer can determine what field they need to improve on like their ball striking, short game, course management, distance, etc.

Use Cases and Fields

The golfer has to download the application through their mobile device or can sign up through the website. After that, the user has to login their personal information and their handicap level. Later, the user's profile will be created in the database and the user can see how they stack up with other players. Afterwards, the player has the option to download a scorecard (stat sheet) that can be used offline to load their data/scores. Once the mobile device has internet connection, it will be automatically loaded into the database.

Field	**What It Stores**	**Why It's Needed**
UserID	Stores a unique number linked to each identity	Just in case a person has a duplicate account or there's another person that has the same name
Email	Stores the email of the user and can also use the email as PK	Only one person is associated with an email
Password	Stores the password to access the account	In order to access the application or website, the user needs a password
FirstName	This stores the first name of the golfer	For golfers to search other users by their first name
LastName	This stores the last name of the golfer	For golfers to search other users by their last name

Handicap	Stores the handicap level of the user	Will be able to compare the user with other users in the same category

Once the fields are completed by the user, their next step is to record their statistics or scores from the rounds they played in 18 holes. This next table shows a few examples of the fields that will be in the database:

Field	**What It Stores**	**Why It's Needed**
Handicap Category	Stores a number (0-36) to show the level of the golfer	The database will store the golfer in a category with other golfers with a similar handicap
Scores	Stores the number of shots a golfer executed in 18 holes	Will use this number to determine a handicap
# of Fairways hit	Will store the # of fairways hit from the tee box (1st shot of a par 4 or 5)	Golfer can use this number to see if they are doing well or poorly off their 1st shot of a hole
# of greens in regulation hit	Will store the # of greens hit in regulation (par 3: if they hit the green on their 1st shot par 4: if they hit the green	This will determine the level of their ball striking and can be compared with others.

	with their second shot, par 5: if they hit the green with their 3rd shot)	
# of Approach	Will store the # of approach shots per hole. Can be a value of zero as well.	This will determine if the user has to focus more on their ball striking if they have a high number of approach shots.
# of Putts	Will store the # of putts per hole.	This will determine if the user is doing well or poorly on the green.
Hole Type	Will store the hole type Par 3, Par 4, or Par 5.	This will determine which type of hole the user is scoring on.

Later the user can input their stats onto the long game table:

Field	**What It Stores**	**Why It's Needed**
Greens Hit from 200-250 yards	Stores the number of greens hit from that distance.	This will determine the accuracy of the player from 200-250 yards out.

Greens Hit from 150-200 yards	Stores the number of greens hit from that distance.	This will determine the accuracy of the player from 150-200 yards out.
Greens Hit from 100-150 yards	Stores the number of greens hit from that distance.	This will determine the accuracy of the player from 100-150 yards out.

To complement the long game, the user can upload their stats into the short game table:

Field	**What It Stores**	**Why It's Needed**
Up and Downs	Stores the number of saves the user has shot during the 18 holes.	This will determine if the user needs to focus their practice their shots around the green.
Sand Saves	Stores the number of sand saves the user has shot during the 18 holes..	This will determine the if the user needs to focus their practice more in their bunker play.

Structural Database Rules

In order to use this free application/website to help the golfer improve their game, they need to follow these steps:

1. User needs to download the application from the app store or can go directly to the website to download the app. Preferably, it will be user-friendly if the golfer downloads the app (quicker to upload the stats into the database).

2. In order to use the application, the user will have to create an account to be able to upload their golf statistics and see how they match-up against others.
3. The golfer will enter their information and the account will be created in our database.

After the user has created an account, they will be able to input their golf handicap and their scores (scores from only 18 holes). Once they input their handicap, the database will automatically place the user into a certain category. From there, the user can see stats from other golfers as well.

Inputting the data into the application will include:

1. Input score from the last 18 holes they played (has to be 18 holes in order to be a validated score)
2. Input the golf course name, city, state, and country where the course was played at.

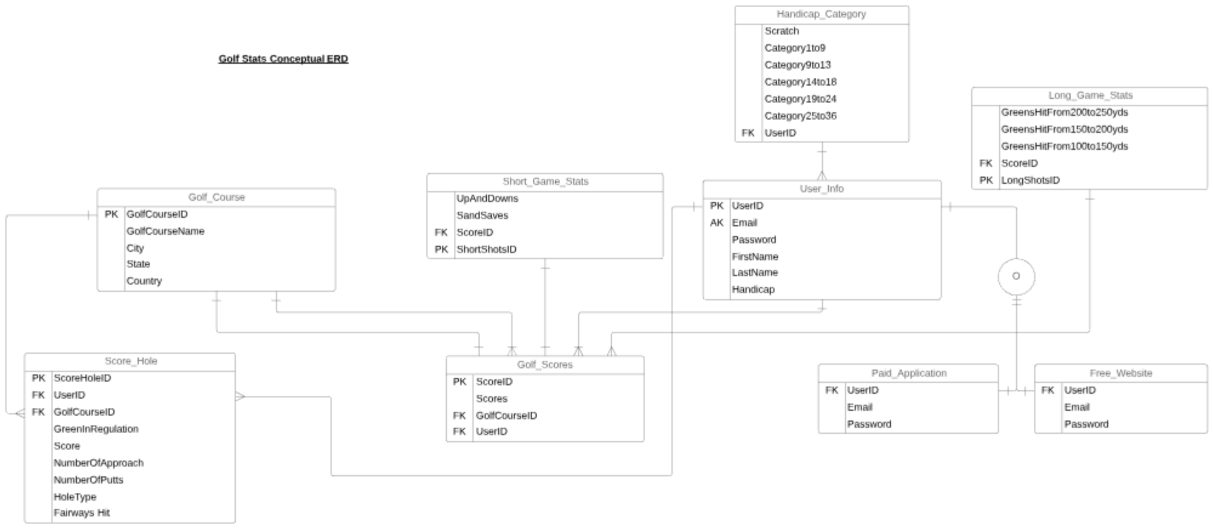
With those 2 steps, there are 2 major data points to store the user's golf score (score and golf course name). After, the database will place the user in the handicap category they stated.

The structural data rules are as follows:

1. A player will submit a score of 18 holes; a score will be submitted by many players.
2. A player will input the difficulty of the golf course; the difficulty of a golf course will be submitted by many players.
3. A player will be placed in a handicap category; each handicap category will have many players.
4. A user has a certain handicap category; a handicap category has many users.
5. A golf course can have many golf scores; a golf score can only be played by a user at a golf course.

6. A user can hit several short game shots; a short game shot is only hit by one user.
7. A user can hit several long game shots; a long game shot is only hit by one user.
8. A golf course can have many par 3s; a par 3 can only be at a golf course.
9. A golf course can have many par 4s; a par 4 can only be at a golf course.
10. A golf course can have many par 5s; a par 5 can only be at a golf course.

Entity – Relationship Diagram



Specialization – Generalization Relationships

(OLD) In order to use this free application/website to help the golfer improve their game, they need to follow these steps:

1. User needs to download the application from the app store or can go directly to the website to download the app. Preferably, it will be user-friendly if the golfer downloads the app (quicker to upload the stats into the database.)

2. In order to use the application, the user will have to create an account to be able to upload their golf statistics and see how they match-up against others.

The golfer will enter their information and the account will be created in our database.

(NEW) In order to use this free application/website to help the golfer improve their game, they need to follow these steps:

1. User can go directly to the website to sign up for free or pay in order to download the app. App will be more user-friendly since it will be quicker to upload the stats into the database.
2. In order to use the application, the user will have to create an account to be able to upload their golf statistics and see how they match-up against others. Same account can be both used in the website and app.

The golfer will enter their information and the account will be created in our database.

Structural Database Rule: An account is a free website service or a paid app.

The super-type would be the User Account and the sub-types would be the application and the website.

The 2 tables I added are: Paid_Application and Free_Website.

Paid_Application:

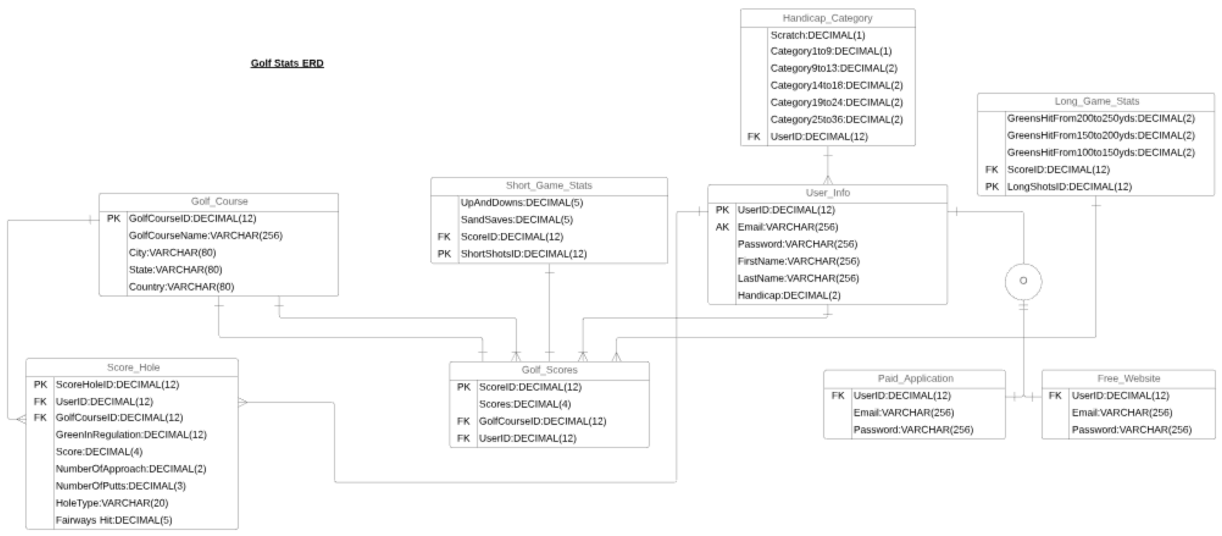
Field	**What It Stores**	**Why It's Needed**
User ID	Stores a unique number linked to each identity	Just in case a person has a duplicate account or there's another person that has the same name
Email	Stores the email of the user and can also use the email as PK	Only one person is associated with an email
Password	Stores the password to access the account	In order to access the application or website, the user needs a password

Free_Website:

Field	**What It Stores**	**Why It's Needed**
User ID	Stores a unique number linked to each identity	Just in case a person has a duplicate account or there's another person that has the same name

Email	Stores the email of the user and can also use the email as PK	Only one person is associated with an email
Password	Stores the password to access the account	In order to access the application or website, the user needs a password

Initial DBMS Physical ERD



Physical ERD

Golf_Course Table:

Attributes	**Datatype**	**Reasoning**
GolfCourseID	BIGINT	Every course has a specific ID because there are some golf courses that have the

		same name in the same city/state/country. This ID will be unique. I chose big integer since an ID can contain many numbers.
GolfCourseName	VARCHAR(256)	Every golf course has a name and I allow up to 256 characters for the name.
City	VARCHAR(80)	The city where the course is located at and I allow up to 80 characters for the city name.
State	VARCHAR(80)	The state where the course is located at and I allow up to 80 characters for the state name.
Country	VARCHAR(80)	The country where the course is located at and I allow up to 80 characters for the country name.

Score_Hole Table:

Attributes	**Datatype**	**Reasoning**
UserID	BIGINT	This will be designated to each user and the number will be unique. I chose big integer since an ID can contain many numbers.
GolfCourseID	BIGINT	Every course has a specific ID because there are some golf courses that have the

		same name in the same city/state/country. I chose big integer since an ID can contain many numbers.
GreenInRegulation	TINYINT	This will record the number of greens hit per round. For Par 3: 1st shot has to land on the green if not it's not counted. For Par 4: 2nd shot has to land on the green if not it's not counted. For Par 5: 3rd shot has to land on the green if not it's not counted. I chose a tiny INT since it is a relatively small number.
Score	TINYINT	Score of 18 holes. I chose a tiny INT since it is a relatively small number.
NumberOfApproaches	TINYINT	This number will determine the number of approaches if the golfer misses the green (how many short game shots the golfer has to hit). I chose a tiny INT since it is a relatively small number.
NumberOfPutts	TINYINT	This number will determine the number of putts per round. I chose a tiny INT since it is a relatively small number.
FairwaysHit	TINYINT	This number will determine the number of fairways hit per round for Par 4's and Par

		5’s. I chose a tiny INT since it is a relatively small number.

Short_Game_Stats Table:

Attributes	**Datatype**	**Reasoning**
UpAndDowns	TINYINT	This will determine the number of Up and Downs per golf round. It is determined by: if the golfer hit an approach to the green and then hit a single putt to finish the hole. If they didn’t, it is not counted as an Up and Down. I chose a tiny INT since it is a relatively small number.
SandSaves	TINYINT	This will determine the number of Sand saves from the bunkers per round. It is determined by: if the golfer hit a greenside bunker shot to the green and then hit a single putt to finish the hole. If they didn’t, it is not counted as a Sand Save. I chose a tiny INT since it is a relatively small number.
ScoreID	BIGINT	This will be designated to each score and the number will be unique since many

		scores can be the same. I chose big integer since an ID can contain many numbers.
ShortShotsID	BIGINT	This will be designated to each Short Shots statistic and the number will be unique since many shots can have the same value. I chose big integer since an ID can contain many numbers.

Long_Game_Stats Table:

Attributes	**Datatype**	**Reasoning**
GreensHitFrom200to250yds	TINYINT	This will determine the number of greens hit from 200-250 yards out. If they miss the green, it will not be counted. I chose a tiny INT since it is a relatively small number.
GreensHitFrom150to200yds	TINYINT	This will determine the number of greens hit from 150-200 yards out. If they miss the green, it will not be counted. I chose a tiny INT since it is a relatively small number.

GreensHitFrom100to150yds	TINYINT	This will determine the number of greens hit from 100-150 yards out. If they miss the green, it will not be counted. I chose a tiny INT since it is a relatively small number.
ScoreID	BIGINT	This will be designated to each score and the number will be unique since many scores can be the same. I chose big integer since an ID can contain many numbers.
LongShotsID	BIGINT	This will be designated to each long shot stats and the number will be unique since many long shots can have the same value. I chose big integer since an ID can contain many numbers.

Golf_Scores Table:

Attributes	**Datatype**	**Reasoning**
ScoreID	BIGINT	This will be designated to each score and the number will be unique since many scores can be the same. I chose big

		integer since an ID can contain many numbers.
Scores	TINYINT	Score of 18 holes. I chose a tiny INT since it is a relatively small number.
DatePlayed	DATE	This will determine the date of the golf round played. Will be inputted as a DATE format.
GolfCourseID	BIGINT	Every course has a specific ID because there are some golf courses that have the same name in the same city/state/country. This number will be unique. I chose big integer since an ID can contain many numbers.
UserID	BIGINT	This will be designated to each user and the number will be unique. I chose big integer since an ID can contain many numbers.

Handicap_Category Table:

Attributes	**Datatype**	**Reasoning**
Scratch	TINYINT	This will determine if the player is a scratch player meaning that their

		handicap is 0 or lower. I chose a tiny INT since it is a relatively small number.
Category1to9	TINYINT	This will determine if the player is in the Category 1 to 9 meaning that their handicap is between 1-9. I chose a tiny INT since it is a relatively small number.
Category9to13	TINYINT	This will determine if the player is in the Category 9 to 13 meaning that their handicap is between 9-13. I chose a tiny INT since it is a relatively small number.
Category14to18	TINYINT	This will determine if the player is in the Category 14 to 18 meaning that their handicap is between 14-18. I chose a tiny INT since it is a relatively small number.
Category19to24	TINYINT	This will determine if the player is in the Category 19 to 24 meaning that their handicap is between 19-24. I chose a tiny INT since it is a relatively small number.
Category25to36	TINYINT	This will determine if the player is in the Category 25 to 36 meaning that their handicap is between 25-36. I chose a

		tiny INT since it is a relatively small number.
UserID	BIGINT	This will be designated to each user and the number will be unique. I chose big integer since an ID can contain many numbers.

User_Info Table:

Attributes	**Datatype**	**Reasoning**
UserID	BIGINT	This will be designated to each user and the number will be unique. I chose big integer since an ID can contain many numbers.
Email	VARCHAR(256)	The email will be used to determine the way the golfer can enter their account (their username). I allow 256 characters to be able to identify the username and this field will be unique.
Password	VARCHAR(256)	Every account has a password. It will be stored in an encrypted text format of a maximum of 256 characters.

FirstName	VARCHAR(256)	Contains the first name of the golfer allowing up to 256 characters.
LastName	VARCHAR(256)	Contains the last name of the golfer allowing up to 256 characters.
Handicap	TINYINT	Contains the number of handicap. I chose a tiny INT since it is a relatively small number.

Paid_Application Table:

Attributes	**Datatype**	**Reasoning**
UserID	BIGINT	This will be designated to each user and the number will be unique. I chose big integer since an ID can contain many numbers.
Email	VARCHAR(256)	The email will be used to determine the way the golfer can enter their account (their username). I allow 256 characters to be able to identify the username and this field will be unique.
Password	VARCHAR(256)	Every account has a password. It will be stored in an encrypted text format of a maximum of 256 characters.

Free_Website Table:

Attributes	**Datatype**	**Reasoning**
UserID	BIGINT	This will be designated to each user and the number will be unique. I chose big integer since an ID can contain many numbers.
Email	VARCHAR(256)	The email will be used to determine the way the golfer can enter their account (their username). I allow 256 characters to be able to identify the username and this field will be unique.
Password	VARCHAR(256)	Every account has a password. It will be stored in an encrypted text format of a maximum of 256 characters.

Handicap_Update:

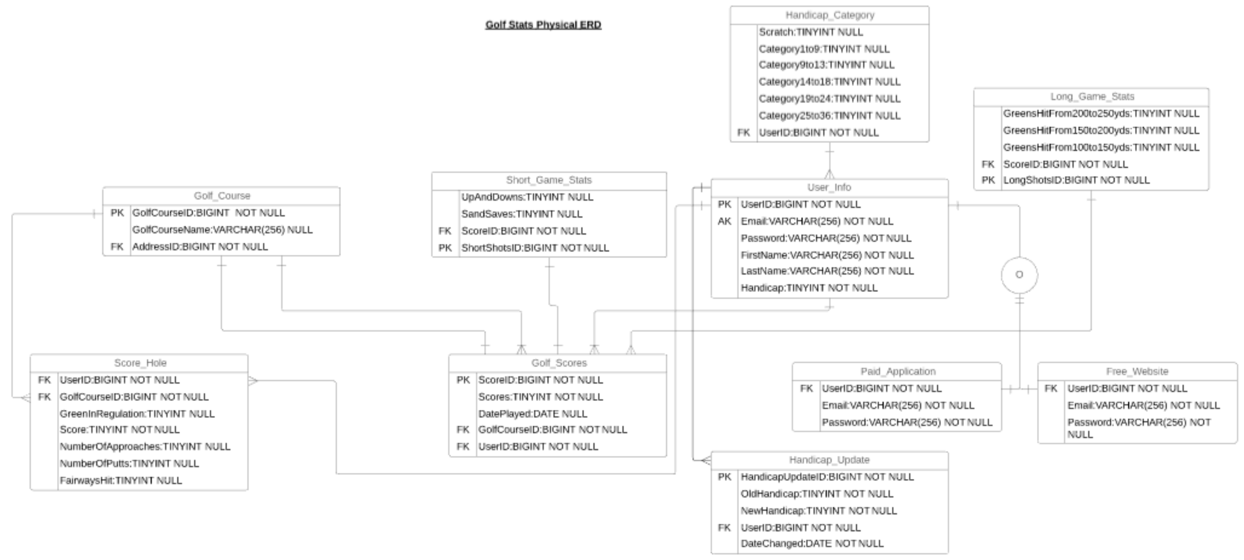
Attributes	**Datatype**	**Reasoning**
HandicapUpdateID	BIGINT	This will be designated to each handicap and the number will be unique. I chose big integer since an ID can contain many numbers.

OldHandicap	TINYINT	Contains the number for the old handicap. I chose a tiny INT since it is a relatively small number.
NewHandicap	TINYINT	Contains the number of the new/updated handicap. I chose a tiny INT since it is a relatively small number.
User ID	BIGINT	This will be designated to each user and the number will be unique. I chose big integer since an ID can contain many numbers.
DateChanged	DATE	Stores the current date of the new handicap that was submitted to the database.

Updated Physical ERD

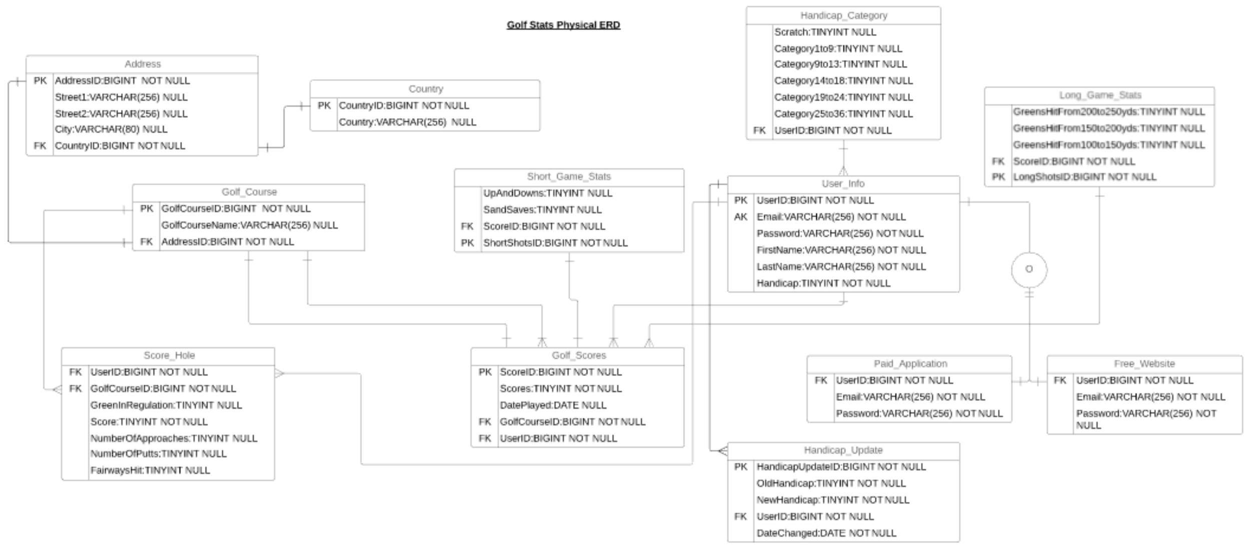
While I was writing my tables down, I had to make some changes in eliminating a couple of attributes that did not make sense with my golf statistic. For instance, both of the attributes were measuring the golf score by hole by hole when in fact, my whole database is measuring the golf score as a whole round of 18 holes. Afterwards, I had to fix the DECIMAL syntaxes since there are no decimal points in golf statistics for a golf round, therefore, I had to use a BIGINT or TINYINT. Based on iteration #5, I also had to update my ERD with adding a new table

“Handicap_Updated” to show the history of the change of handicap per user, so the updated Physical ERD is shown below:


Golf Stats Physical ERD
Handicap_Category
Scratch:TINYINT NULL
Category1to9:TINYINT NULL
Category9to13:TINYINT NULL
Category14to18:TINYINT NULL
Category19to24:TINYINT NULL
Category25to36:TINYINT NULL
FK UserID:BIGINT NOT NULL
Long_Game_Stats
GreensHitFrom200to250yds:TINYINT NULL
GreensHitFrom150to200yds:TINYINT NULL
GreensHitFrom100to150yds:TINYINT NULL
FK ScoreID:BIGINT NOT NULL
PK LongShotsID:BIGINT NOT NULL
Golf_Course
PK GolfCourseID:BIGINT NOT NULL
GolfCourseName:VARCHAR(256) NULL
FK AddressID:BIGINT NOT NULL
Short_Game_Stats
UpAndDowns:TINYINT NULL
SandSaves:TINYINT NULL
FK ScoreID:BIGINT NOT NULL
PK ShortShotsID:BIGINT NOT NULL
User_Info
PK UserID:BIGINT NOT NULL
AK Email:VARCHAR(256) NOT NULL
Password:VARCHAR(256) NOT NULL
FirstName:VARCHAR(256) NOT NULL
LastName:VARCHAR(256) NOT NULL
Handicap:TINYINT NOT NULL
O
Score_Hole
FK UserID:BIGINT NOT NULL
FK GolfCourseID:BIGINT NOT NULL
GreenInRegulation:TINYINT NULL
Score:TINYINT NOT NULL
NumberOfApproaches:TINYINT NULL
NumberOfPutts:TINYINT NULL
FairwaysHit:TINYINT NULL
Golf_Scores
PK ScoreID:BIGINT NOT NULL
Scores:TINYINT NOT NULL
DatePlayed:DATE NULL
FK GolfCourseID:BIGINT NOT NULL
FK UserID:BIGINT NOT NULL
Paid_Application
FK UserID:BIGINT NOT NULL
Email:VARCHAR(256) NOT NULL
Password:VARCHAR(256) NOT NULL
Free_Website
FK UserID:BIGINT NOT NULL
Email:VARCHAR(256) NOT NULL
Password:VARCHAR(256) NOT NULL
Handicap_Update
PK HandicapUpdateID:BIGINT NOT NULL
OldHandicap:TINYINT NOT NULL
NewHandicap:TINYINT NOT NULL
FK UserID:BIGINT NOT NULL
DateChanged:DATE NOT NULL

Normalization

The only section I see where there is redundancy in my physical ERD are the golf course addresses. If many golfers play golf on the same golf course, the address of that golf course will be repetitive. The updated ERD (shown below) is normalized:


Golf Stats Physical ERD
Address
PK AddressID:BIGINT NOT NULL
Street1:VARCHAR(256) NULL
Street2:VARCHAR(256) NULL
City:VARCHAR(80) NULL
FK CountryID:BIGINT NOT NULL
Country
PK CountryID:BIGINT NOT NULL
Country:VARCHAR(256) NULL
Handicap_Category
Scratch:TINYINT NULL
Category1to9:TINYINT NULL
Category9to13:TINYINT NULL
Category14to18:TINYINT NULL
Category19to24:TINYINT NULL
Category25to36:TINYINT NULL
FK UserID:BIGINT NOT NULL
Long_Game_Stats
GreensHitFrom200to250yds:TINYINT NULL
GreensHitFrom150to200yds:TINYINT NULL
GreensHitFrom100to150yds:TINYINT NULL
FK ScoreID:BIGINT NOT NULL
PK LongShotsID:BIGINT NOT NULL
Golf_Course
PK GolfCourseID:BIGINT NOT NULL
GolfCourseName:VARCHAR(256) NULL
FK AddressID:BIGINT NOT NULL
Short_Game_Stats
UpAndDowns:TINYINT NULL
SandSaves:TINYINT NULL
FK ScoreID:BIGINT NOT NULL
PK ShortShotsID:BIGINT NOT NULL
User_Info
PK UserID:BIGINT NOT NULL
AK Email:VARCHAR(256) NOT NULL
Password:VARCHAR(256) NOT NULL
FirstName:VARCHAR(256) NOT NULL
LastName:VARCHAR(256) NOT NULL
Handicap:TINYINT NOT NULL
O
Score_Hole
FK UserID:BIGINT NOT NULL
FK GolfCourseID:BIGINT NOT NULL
GreenInRegulation:TINYINT NULL
Score:TINYINT NOT NULL
NumberOfApproaches:TINYINT NULL
NumberOfPutts:TINYINT NULL
FairwaysHit:TINYINT NULL
Golf_Scores
PK ScoreID:BIGINT NOT NULL
Scores:TINYINT NOT NULL
DatePlayed:DATE NULL
FK GolfCourseID:BIGINT NOT NULL
FK UserID:BIGINT NOT NULL
Paid_Application
FK UserID:BIGINT NOT NULL
Email:VARCHAR(256) NOT NULL
Password:VARCHAR(256) NOT NULL
Free_Website
FK UserID:BIGINT NOT NULL
Email:VARCHAR(256) NOT NULL
Password:VARCHAR(256) NOT NULL
Handicap_Update
PK HandicapUpdateID:BIGINT NOT NULL
OldHandicap:TINYINT NOT NULL
NewHandicap:TINYINT NOT NULL
FK UserID:BIGINT NOT NULL
DateChanged:DATE NOT NULL

The updated structural data rules are as follows:

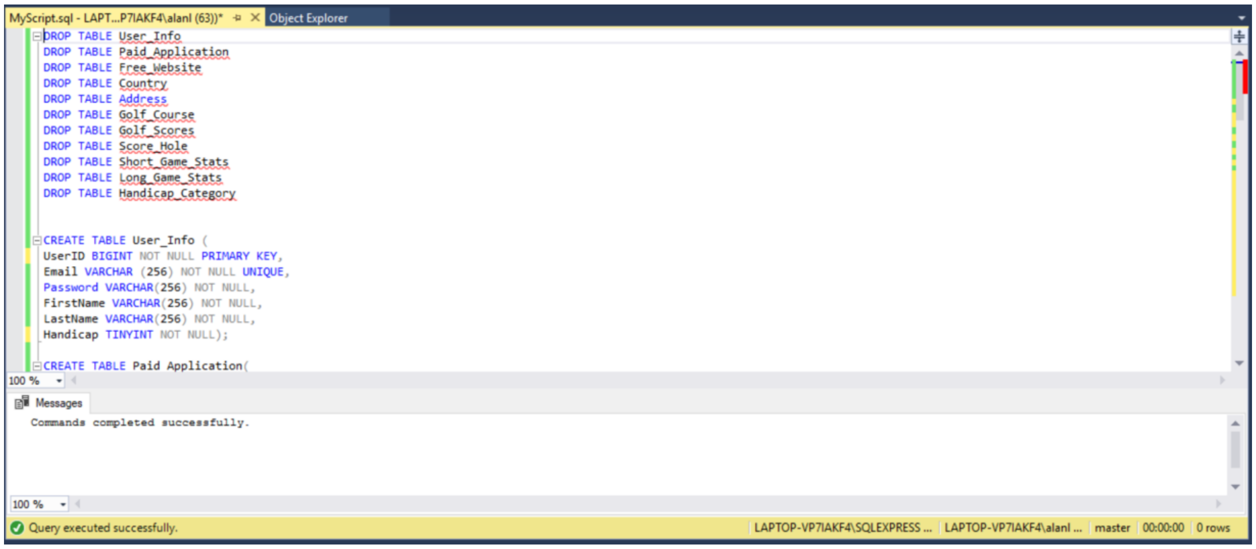
1. A player will submit a score of 18 holes; a score will be submitted by many players.
2. A player will input their handicap category; each handicap category will have many players.
3. A user has a certain handicap category; a handicap category has many users.
4. A golf course can have many golf scores; a golf score can only be played by a user at a golf course.
5. A user can hit several short game shots; a short game shot is only hit by one user.
6. A user can hit several long game shots; a long game shot is only hit by one user.
7. Each golf score is associated to a golf course address; each golf course address can have one or many golf scores.
8. Each golf course address is associated to a country; each country has many golf course addresses.
9. Each user can have many handicap updates; each handicap update is for one user.

Entity Check

- If we know the email of the golfer, we are able to determine the first name, last name, and the password of the golfer.
- If we only know the first name or last name of the golfer, we cannot determine the email of the golfer.
- If we only know the password of the golfer we cannot determine the first name, last name, or email of the golfer.

- If the golfer inputs the golf course address, then we can determine the country of the golf course.
- If the golfer only inputs the country, we will not be able to determine which golf course it is because there might be golf courses that have the same name in the same country.
- There can be many data entries of the same golf score, but each golf score will have its unique ID alongside with their stats.
- If the golfer does not input a score, then the golfer cannot input their other golf stats.
- A golfer must input a handicap to the database so the database can determine which category the golfer will be placed at.
- If the golfer does not input a handicap to the database, then the golfer will not be able to compare themselves with other golfers within the same handicap category they are in.
- A golfer must input a score of 18 holes in order to determine the statistics per round.
- If a golfer does not input a score of 18 holes, then the golf statistics will be skewed and inaccurate.

Tables Created

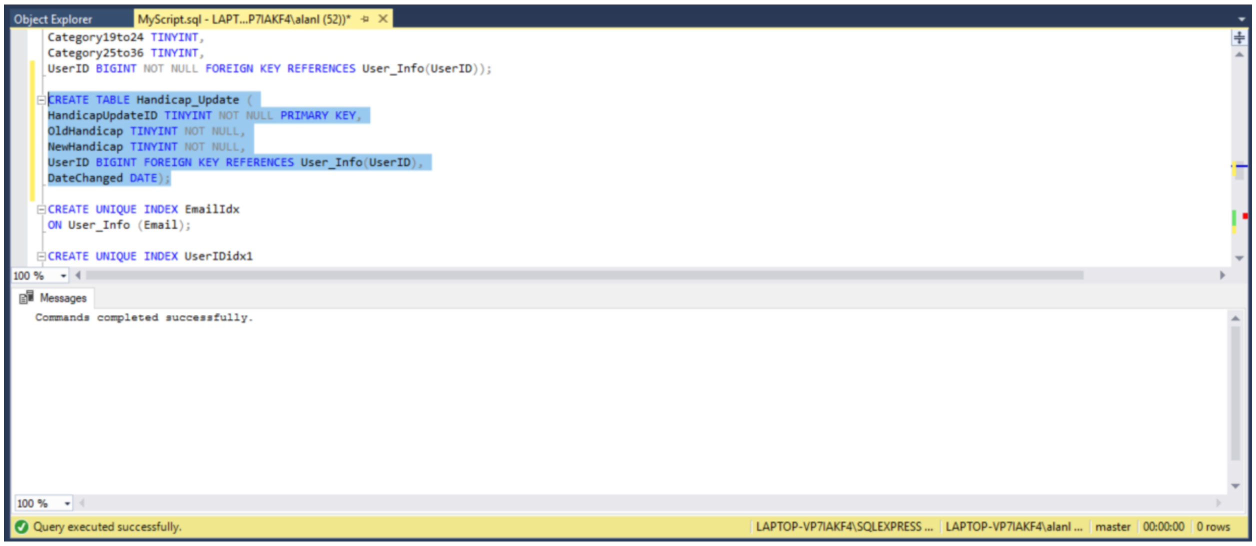


```
DROP TABLE User_Info
DROP TABLE Paid_Application
DROP TABLE Free_Website
DROP TABLE Country
DROP TABLE Address
DROP TABLE Golf_Course
DROP TABLE Golf_Scores
DROP TABLE Score_Hole
DROP TABLE Short_Game_Stats
DROP TABLE Long_Game_Stats
DROP TABLE Handicap_Category


CREATE TABLE User_Info (
UserID BIGINT NOT NULL PRIMARY KEY,
Email VARCHAR (256) NOT NULL UNIQUE,
Password VARCHAR(256) NOT NULL,
FirstName VARCHAR(256) NOT NULL,
LastName VARCHAR(256) NOT NULL,
Handicap TINYINT NOT NULL);

CREATE TABLE Paid Application(
```

```
Commands completed successfully.
```

```
Query executed successfully.    LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 0 rows
```

I updated my ERD with a new table: “Handicap_Update” to show the user the history of when their handicap changed. Shown below is the screenshot of the addition table created.



```
Category19to24 TINYINT,
Category25to36 TINYINT,
UserID BIGINT NOT NULL FOREIGN KEY REFERENCES User_Info(UserID));

CREATE TABLE Handicap_Update (
HandicapUpdateID TINYINT NOT NULL PRIMARY KEY,
OldHandicap TINYINT NOT NULL,
NewHandicap TINYINT NOT NULL,
UserID BIGINT FOREIGN KEY REFERENCES User_Info(UserID),
DateChanged DATE);

CREATE UNIQUE INDEX EmailIdx
ON User_Info (Email);

CREATE UNIQUE INDEX UserIDidx1
```

```
Commands completed successfully.
```

```
Query executed successfully.    LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 0 rows
```

Index Placement

Primary Keys:

Primary Key Column	**Description**
User_Info.UserID	User ID is the primary key of the User_Info table.
User_Info.Email	Email is also the primary key of the User_Info table.
Golf_Scores.Score_ID	Score_ID is the priamary key of the Golf_Scores table.

Golf_Course.GolfCourseID	GolfCourseID is the primary key of the Golf _Course table.
Address.AddressID	AddressID is the primary key of the Address table.
Country.CountryID	CountryID is the primary key of the Country table.
Short_Game_Stats.ShortShotsID	ShortShotsID is the primary key of the Short_Game_Stats table.
Long_Game_Stats.LongShotsID	LongShotsID is the primary key of the Long_Game_Stats table.
HandicapUpdateID	HandicapUpdateID is the primary key of the Handicap_Update table.

Foreign Keys Indexed:

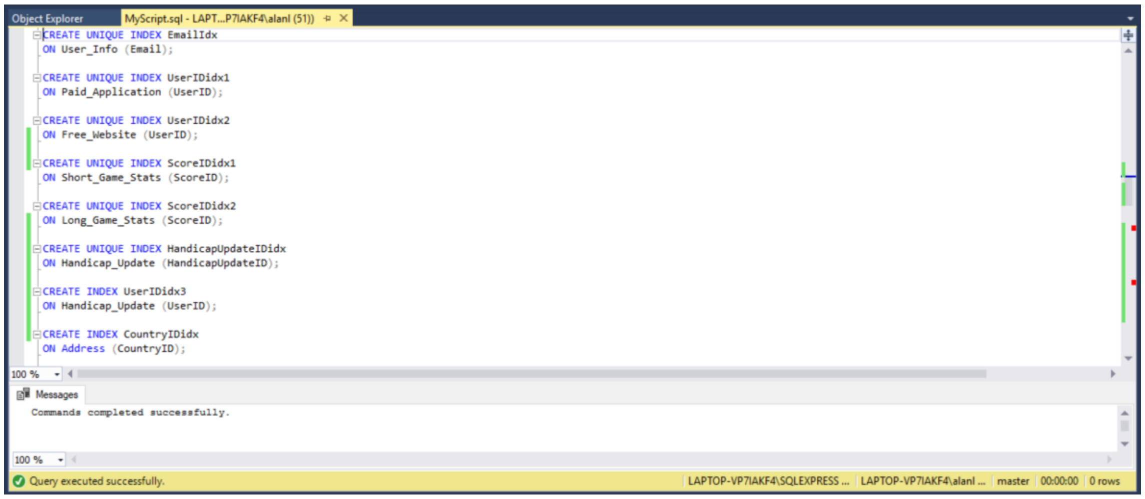
Foreign Key Column	**Unique?**	**Description**
User_Info.Email	Unique	Email also acts as a foreign key that's why it is an alternate key. It is a unique because only one person can have a certain email and cannot be redundant.
Paid_Application.UserID	Unique	UserID is a foreign key to Paid_Application table since it

		references the primary key of the User_Info table. It is unique because a user can only have a paid application.
Free_Website.UserID	Unique	UserID is a foreign key to Free_Website table since it references the primary key of the User_Info table. It is unique because a user can only have a free website account.
Address.CountryID	Non-Unique	CountryID is a foreign key to the Address table since it references the primary key of the Country table and it’s not unique since many countries have many addresses.
Golf_Course.AddressID	Non-Unique	AddressID is a foreign key to the Golf_Course table since it references the primary key of Address table and it’s not unique since many golf courses have many addresses.

Golf_Scores.GolfCourseID	Non-Unique	GolfCourseID is a foreign key to the Golf_Scores table since it references the primary key of the Golf_Course table and it’s not unique because a golf course can have many golf scores.
Golf_Scores.UserID	Non-Unique	UserID is a foreign key to Golf_Scores table since it references the primary key of the User_Info table. It is not unique because a user can have many golf scores.
Score_Hole.UserID	Non-Unique	UserID is a foreign key to Score_Hole table since it references the primary key of the User_Info table. It is not unique because a user can have many statistics per score.
Score_Hole.GolfCourseID	Non-Unique	GolfCourseID is a foreign key to the Score_Hole table since it references the primary key of the Golf_Course table and it’s

		not unique because a golf course can have many golf statistics per score.
Short_Game_Stats.ScoreID	Unique	ScoreID is a foreign key to Short_Game_Stats table since it references the primary key of the Golf_Scores table and it is unique because a score will have statistics for short game shots.
Long_Game_Stats.ScoreID	Unique	ScoreID is a foreign key to Long_Game_Stats table since it references the primary key of the Golf_Scores table and it is unique because a score will have statistics for long game shots.
Handicap_Update.UserID	Non-Unique	UserID is a foreign key to Handicap _Update table since it references the primary key of the User_Info table. It is not unique because a user can have many handicap changes.

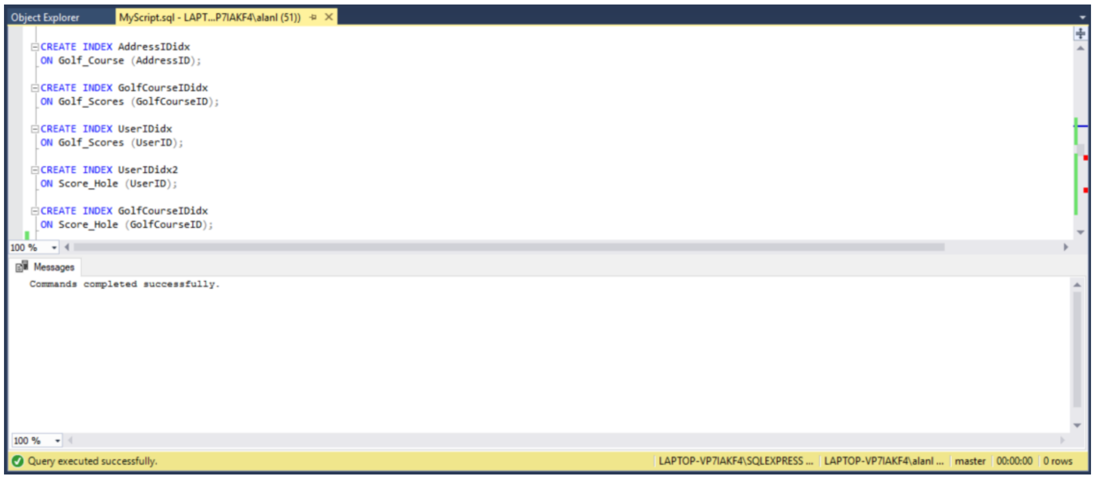
Creating Indexes in SQL



```
CREATE UNIQUE INDEX EmailIdx
ON User_Info (Email);

CREATE UNIQUE INDEX UserIDidx1
ON Paid_Application (UserID);

CREATE UNIQUE INDEX UserIDidx2
ON Free_Website (UserID);

CREATE UNIQUE INDEX ScoreIDidx1
ON Short_Game_Stats (ScoreID);

CREATE UNIQUE INDEX ScoreIDidx2
ON Long_Game_Stats (ScoreID);

CREATE UNIQUE INDEX HandicapUpdateIDidx
ON Handicap_Update (HandicapUpdateID);

CREATE INDEX UserIDidx3
ON Handicap_Update (UserID);

CREATE INDEX CountryIDidx
ON Address (CountryID);
```

```
Commands completed successfully.
```

Query executed successfully. | LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 0 rows



```
CREATE INDEX AddressIDidx
ON Golf_Course (AddressID);

CREATE INDEX GolfCourseIDidx
ON Golf_Scores (GolfCourseID);

CREATE INDEX UserIDidx
ON Golf_Scores (UserID);

CREATE INDEX UserIDidx2
ON Score_Hole (UserID);

CREATE INDEX GolfCourseIDidx
ON Score_Hole (GolfCourseID);
```

```
Commands completed successfully.
```

Query executed successfully. | LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 0 rows

CREATE UNIQUE INDEX EmailIdx
ON User_Info (Email);

I named the index "EmailIdx" to help identify the email uniqueness and place it in the Email column in User_Info.

CREATE UNIQUE INDEX UserIDidx1
ON Paid_Application (UserID);

I named the index "UserIDidx1" to help identify the user uniqueness and place it in the UserID column in Paid_Application.

CREATE UNIQUE INDEX UserIDidx2
ON Free_Website (UserID);

I named the index "UserIDidx2" to help identify the user uniqueness and place it in the UserID column in Free_Website.

```
CREATE UNIQUE INDEX ScoreIDidx1
ON Short_Game_Stats (ScoreID);
```

I named the index "ScoreIDidx1" to help identify the score uniqueness and place it in the ScoreID column in Short_Game_Stats.

```
CREATE UNIQUE INDEX ScoreIDidx2
ON Long_Game_Stats (ScoreID);
```

I named the index "ScoreIDidx2" to help identify the score uniqueness and place it in the ScoreID column in Long_Game_Stats.

```
CREATE UNIQUE INDEX HandicapUpdateIDidx
ON Handicap_Update (HandicapUpdateID);
```

I named the index "HandicapUpdateIDidx" to help identify the handicap uniqueness and place it in the HandicapUpdateID column in Handicap_Update table.

```
CREATE INDEX UserIDidx3
ON Handicap_Update (UserID);
```

I named the index "UserIDidx3" to help identify the handicap ID and place it in the UserID column in the Handicap_Update table.

```
CREATE INDEX CountryIDidx
ON Address (CountryID);
```

I named the index "CountryIDidx" to help identify the country ID and place it in the CountryID column in Address.

```
CREATE INDEX AddressIDidx
ON Golf_Course (AddressID);
```

I named the index "AddressIDidx" to help identify the Address ID and place it in the AddressID column in Golf_Course.

```
CREATE INDEX GolfCourseIDidx
ON Golf_Scores (GolfCourseID);
```

I named the index "GolfCourseIDidx" to help identify the Golf Course ID and place it in the GolfCourseID column in Golf_Scores.

```
CREATE INDEX UserIDidx
ON Golf_Scores (UserID);
```

I named the index "UserIDidx" to help identify the User ID and place it in the UserID column in Golf_Scores.

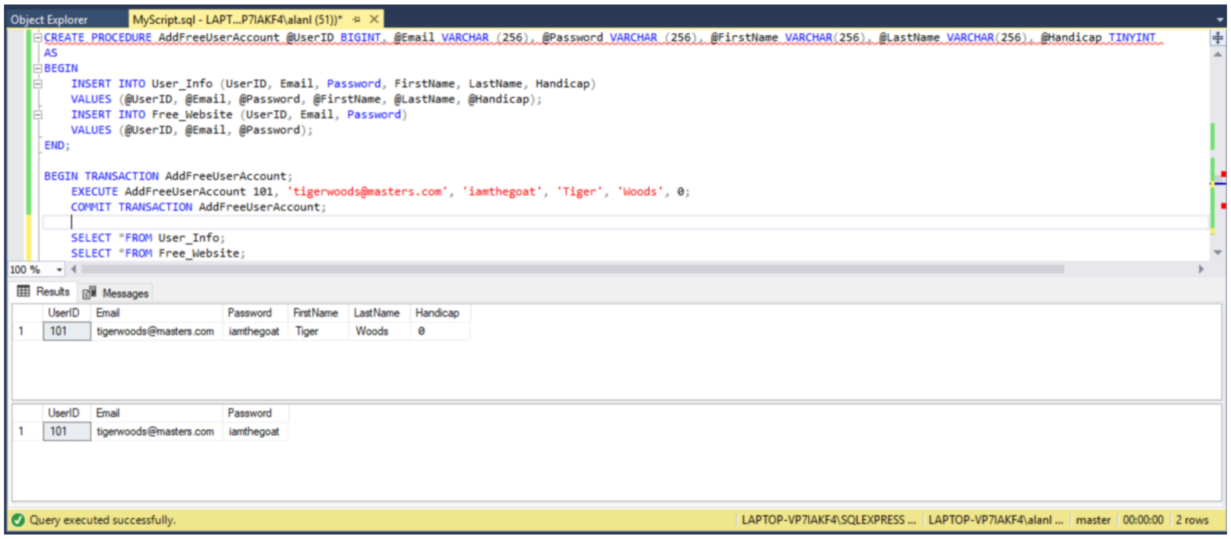
```
CREATE INDEX UserIDidx2
ON Score_Hole (UserID);
```

I named the index "UserIDidx2" to help identify the User ID and place it in the UserID column in Score_Hole.

```
CREATE INDEX GolfCourseIDidx
ON Score_Hole (GolfCourseID);
```

I named the index "GolfCourseIDidx" to help identify the Golf Course ID and place it in the GolfCourseID column in Score_Hole.

Procedures

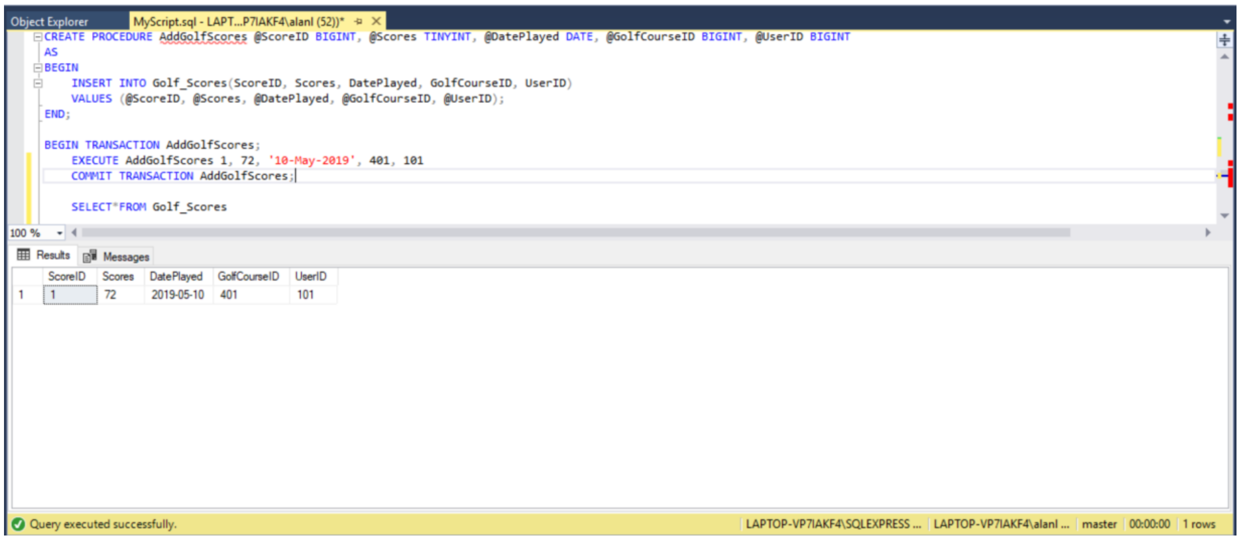


```
CREATE PROCEDURE AddFreeUserAccount @UserID BIGINT, @Email VARCHAR (256), @Password VARCHAR (256), @FirstName VARCHAR(256), @LastName VARCHAR(256), @Handicap TINYINT
AS
BEGIN
    INSERT INTO User_Info (UserID, Email, Password, FirstName, LastName, Handicap)
    VALUES (@UserID, @Email, @Password, @FirstName, @LastName, @Handicap);
    INSERT INTO Free_Website (UserID, Email, Password)
    VALUES (@UserID, @Email, @Password);
END;

BEGIN TRANSACTION AddFreeUserAccount;
    EXECUTE AddFreeUserAccount 101, 'tigerwoods@masters.com', 'iamthegoat', 'Tiger', 'Woods', 0;
    COMMIT TRANSACTION AddFreeUserAccount;

    SELECT *FROM User_Info;
    SELECT *FROM Free_Website;
```

	UserID	Email	Password	FirstName	LastName	Handicap
1	101	tigerwoods@masters.com	iamthegoat	Tiger	Woods	0

	UserID	Email	Password
1	101	tigerwoods@masters.com	iamthegoat

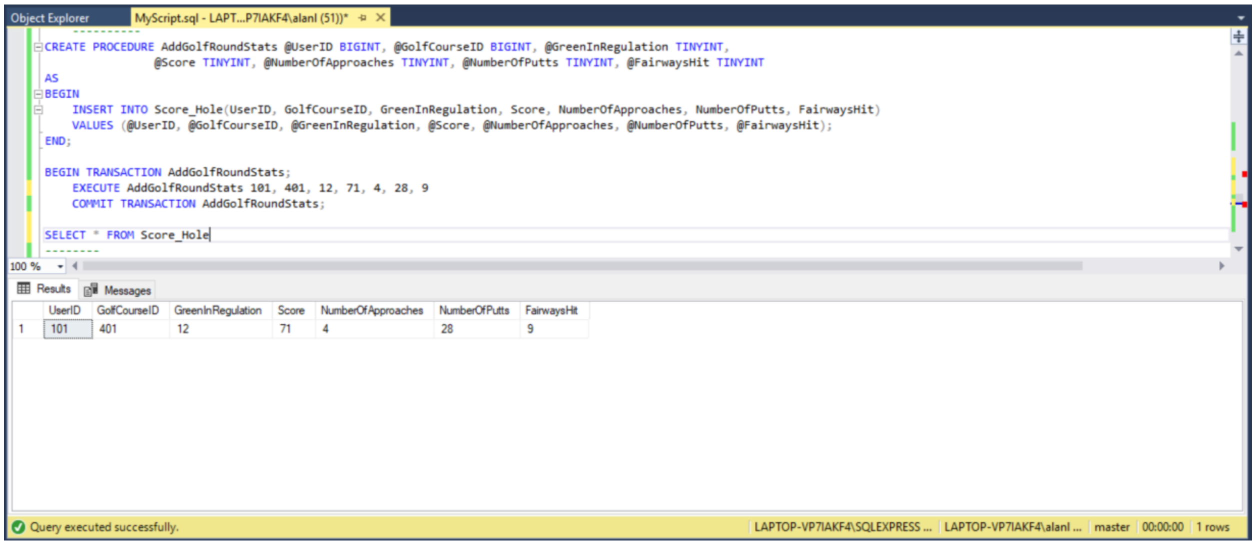
```
Query executed successfully.    LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 2 rows
```



```sql
CREATE PROCEDURE AddGolfScores @ScoreID BIGINT, @Scores TINYINT, @DatePlayed DATE, @GolfCourseID BIGINT, @UserID BIGINT
AS
BEGIN
    INSERT INTO Golf_Scores(ScoreID, Scores, DatePlayed, GolfCourseID, UserID)
    VALUES (@ScoreID, @Scores, @DatePlayed, @GolfCourseID, @UserID);
END;

BEGIN TRANSACTION AddGolfScores;
    EXECUTE AddGolfScores 1, 72, '10-May-2019', 401, 101
    COMMIT TRANSACTION AddGolfScores;

    SELECT*FROM Golf_Scores
```

	ScoreID	Scores	DatePlayed	GolfCourseID	UserID
1	1	72	2019-05-10	401	101

```
Query executed successfully.    LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 1 rows
```

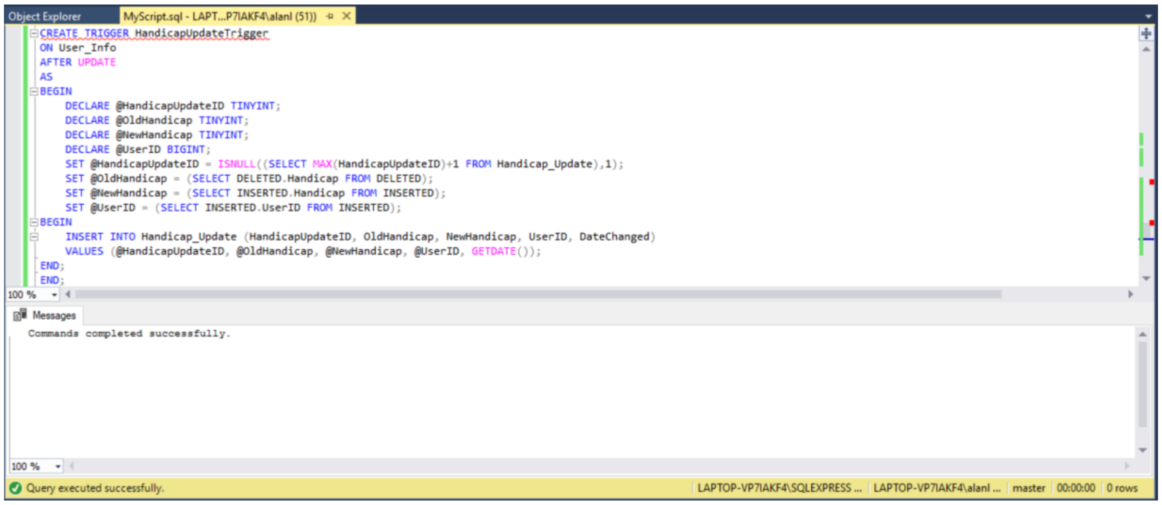


```sql
CREATE PROCEDURE AddGolfRoundStats @UserID BIGINT, @GolfCourseID BIGINT, @GreenInRegulation TINYINT,
                 @Score TINYINT, @NumberOfApproaches TINYINT, @NumberOfPutts TINYINT, @FairwaysHit TINYINT
AS
BEGIN
    INSERT INTO Score_Hole(UserID, GolfCourseID, GreenInRegulation, Score, NumberOfApproaches, NumberOfPutts, FairwaysHit)
    VALUES (@UserID, @GolfCourseID, @GreenInRegulation, @Score, @NumberOfApproaches, @NumberOfPutts, @FairwaysHit);
END;

BEGIN TRANSACTION AddGolfRoundStats;
    EXECUTE AddGolfRoundStats 101, 401, 12, 71, 4, 28, 9
    COMMIT TRANSACTION AddGolfRoundStats;

SELECT * FROM Score_Hole
```

	UserID	GolfCourseID	GreenInRegulation	Score	NumberOfApproaches	NumberOfPutts	FairwaysHit
1	101	401	12	71	4	28	9

```
Query executed successfully.    LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 1 rows
```

Updating Handicap Level Through Trigger



```
CREATE TRIGGER HandicapUpdateTrigger
ON User_Info
AFTER UPDATE
AS
BEGIN
    DECLARE @HandicapUpdateID TINYINT;
    DECLARE @OldHandicap TINYINT;
    DECLARE @NewHandicap TINYINT;
    DECLARE @UserID BIGINT;
    SET @HandicapUpdateID = ISNULL((SELECT MAX(HandicapUpdateID)+1 FROM Handicap_Update),1);
    SET @OldHandicap = (SELECT DELETED.Handicap FROM DELETED);
    SET @NewHandicap = (SELECT INSERTED.Handicap FROM INSERTED);
    SET @UserID = (SELECT INSERTED.UserID FROM INSERTED);
BEGIN
    INSERT INTO Handicap_Update (HandicapUpdateID, OldHandicap, NewHandicap, UserID, DateChanged)
    VALUES (@HandicapUpdateID, @OldHandicap, @NewHandicap, @UserID, GETDATE());
END;
END;
```

```
Commands completed successfully.
```

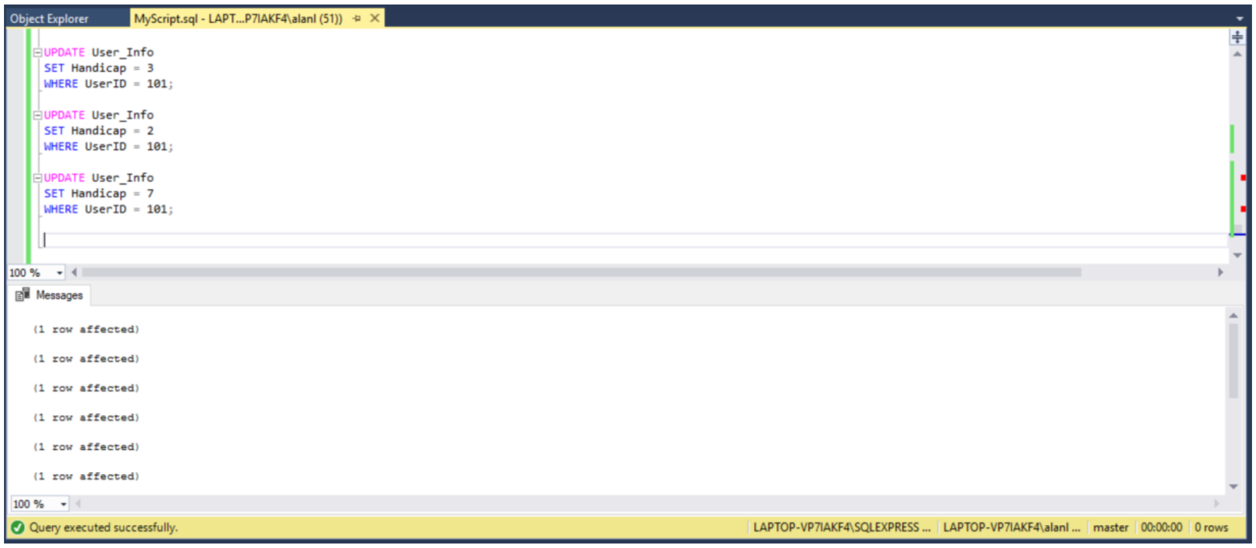
Query executed successfully. | LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 0 rows

Code	Description
`CREATE TRIGGER HandicapUpdateTrigger` `ON User_Info`	This names the trigger "HandicapUpdateTrigger" and links it to the User_Info table.
`AFTER UPDATE` `AS` `BEGIN`	This indicates that the trigger should run after the table (ignoring INSERTS and DELETES), along with some keywords needed by the T-SQL syntax.
`DECLARE @HandicapUpdateID TINYINT;` `DECLARE @OldHandicap TINYINT;` `DECLARE @NewHandicap TINYINT;` `DECLARE @UserID BIGINT;`	Captures the Handicap Update ID, old handicap, new handicap, and user ID by accessing the DELETED and INSERTED pseudo tables provided by T-SQL.
`SET @HandicapUpdateID = ISNULL((SELECT MAX(HandicapUpdateID)+1 FROM Handicap_Update),1);`	The Handicap Update ID is generated by getting the maximum HandicapUpdateID + 1 if one exists, or 1 otherwise. The old handicap

`SET @OldHandicap = (SELECT DELETED.Handicap FROM DELETED);`
`SET @NewHandicap = (SELECT INSERTED.Handicap FROM INSERTED);`
`SET @UserID = (SELECT INSERTED.UserID FROM INSERTED);`
	will be replaced by the new handicap. The UserID will be extracted from the INSERTED pseudo table provided by T-SQL.

`BEGIN`
`INSERT INTO Handicap_Update (HandicapUpdateID, OldHandicap, NewHandicap, UserID, DateChanged)`
`VALUES (@HandicapUpdateID, @OldHandicap, @NewHandicap, @UserID, GETDATE());`
	This command will begin by inserting the values (after there is an update). The order will be the HandicapUpdateID (which will be automatically generated by the max value + 1 or if there's no value then 1), OldHandicap (the orginal/past handicap value), NewHandicap (current handicap value), UserID (user's ID), GETDATE (used to obtain the current date).

`END;` `END;`	This ends the trigger definition.



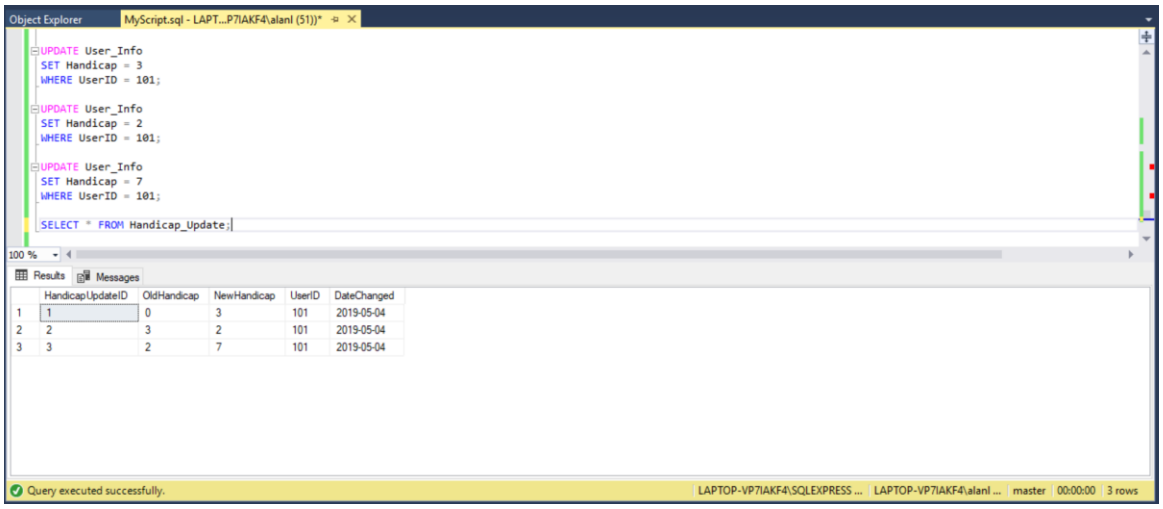
Object Explorer | MyScript.sql - LAPT...P7IAKF4\alanl (51))

```
UPDATE User_Info
SET Handicap = 3
WHERE UserID = 101;

UPDATE User_Info
SET Handicap = 2
WHERE UserID = 101;

UPDATE User_Info
SET Handicap = 7
WHERE UserID = 101;
```

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

Query executed successfully. | LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 0 rows



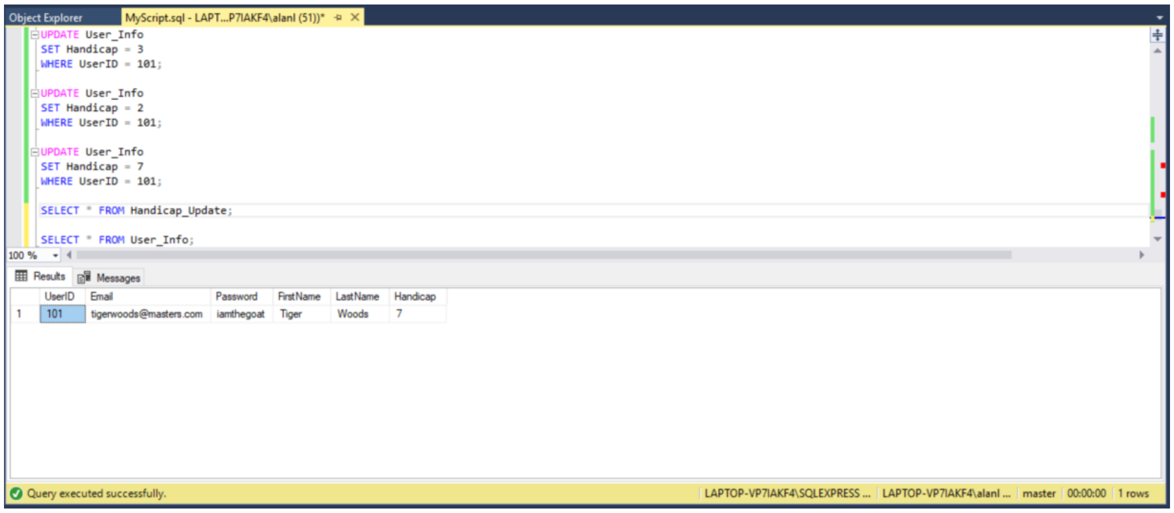
Object Explorer | MyScript.sql - LAPT...P7IAKF4\alanl (51))*

```
UPDATE User_Info
SET Handicap = 3
WHERE UserID = 101;

UPDATE User_Info
SET Handicap = 2
WHERE UserID = 101;

UPDATE User_Info
SET Handicap = 7
WHERE UserID = 101;

SELECT * FROM Handicap_Update;
```

Results | Messages

	HandicapUpdateID	OldHandicap	NewHandicap	UserID	DateChanged
1	1	0	3	101	2019-05-04
2	2	3	2	101	2019-05-04
3	3	2	7	101	2019-05-04

Query executed successfully. | LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 3 rows



Object Explorer | MyScript.sql - LAPT...P7IAKF4\alanl (51))*

```
UPDATE User_Info
SET Handicap = 3
WHERE UserID = 101;

UPDATE User_Info
SET Handicap = 2
WHERE UserID = 101;

UPDATE User_Info
SET Handicap = 7
WHERE UserID = 101;

SELECT * FROM Handicap_Update;

SELECT * FROM User_Info;
```

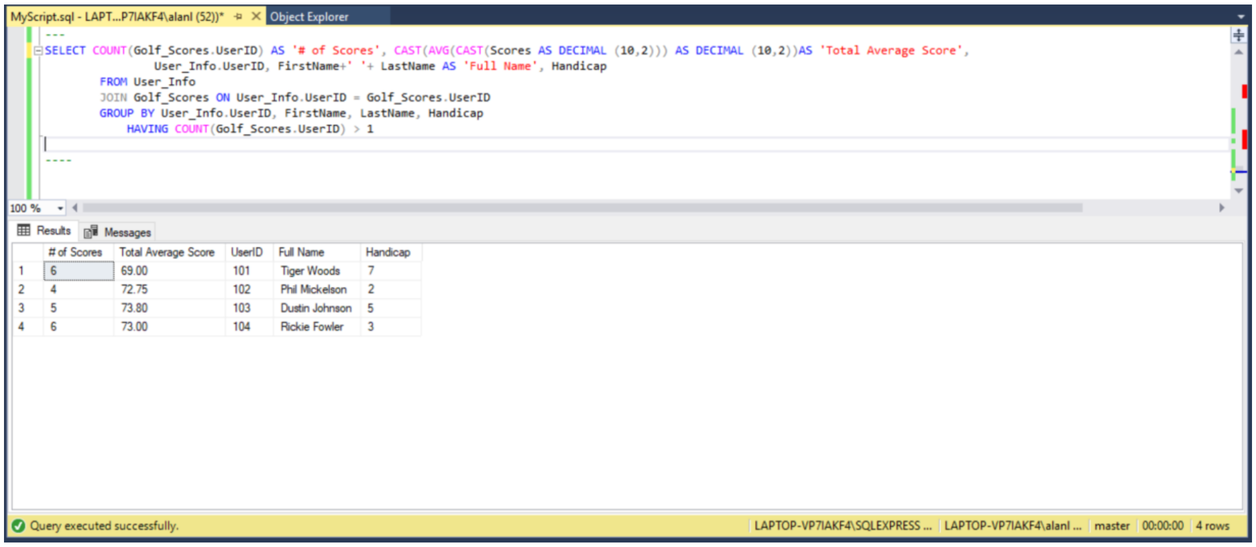
Results | Messages

	UserID	Email	Password	FirstName	LastName	Handicap
1	101	tigerwoods@masters.com	iamthegoat	Tiger	Woods	7

Query executed successfully. | LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 1 rows

Creating Questions and Queries for the Database

I wanted to create a question to show the average scores of all the golf players in my database. The player has to have submitted more than one golf round (18 holes) in order to be qualified for this query. "Jonathan Lee" was excluded in the query since he only played one golf round of 90 shots. The query below will demonsrate the user the total average scores per player.



MyScript.sql - LAPT...P7IAKF4\alanl (52))*   Object Explorer

```
---
SELECT COUNT(Golf_Scores.UserID) AS '# of Scores', CAST(AVG(CAST(Scores AS DECIMAL (10,2))) AS DECIMAL (10,2))AS 'Total Average Score',
                User_Info.UserID, FirstName+' '+ LastName AS 'Full Name', Handicap
        FROM User_Info
        JOIN Golf_Scores ON User_Info.UserID = Golf_Scores.UserID
        GROUP BY User_Info.UserID, FirstName, LastName, Handicap
            HAVING COUNT(Golf_Scores.UserID) > 1

----
```

100 %

Results   Messages

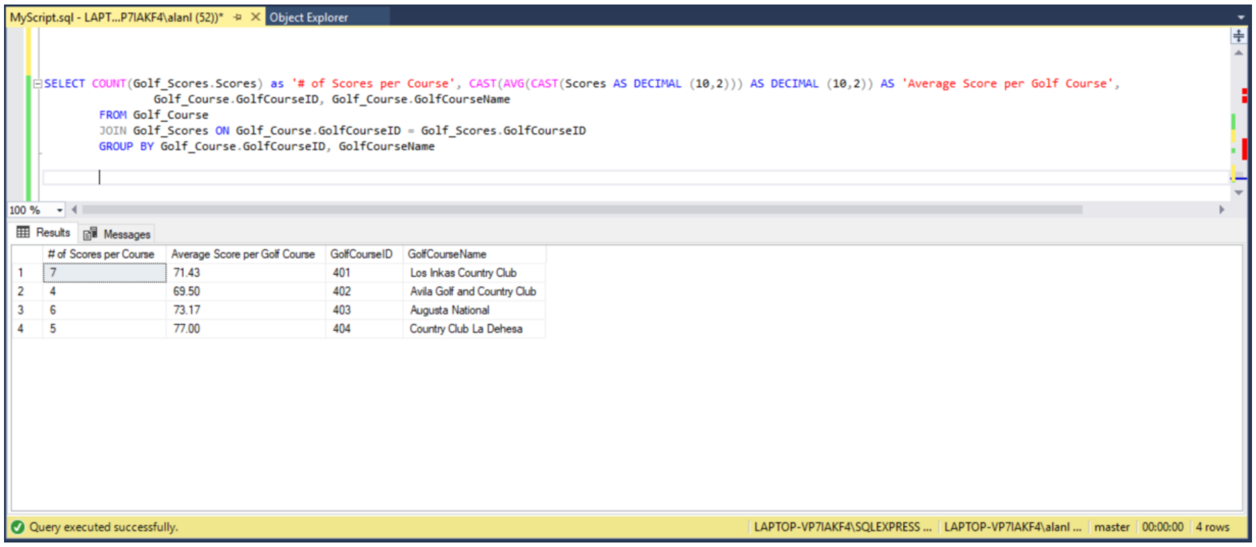
	# of Scores	Total Average Score	UserID	Full Name	Handicap
1	6	69.00	101	Tiger Woods	7
2	4	72.75	102	Phil Mickelson	2
3	5	73.80	103	Dustin Johnson	5
4	6	73.00	104	Rickie Fowler	3

Query executed successfully.   LAPTOP-VP7IAKF4\SQLEXPRESS ...   LAPTOP-VP7IAKF4\alanl ...   master   00:00:00   4 rows

Code	Description
`SELECT COUNT(Golf_Scores.UserID) AS '# of Scores', CAST(AVG(CAST(Scores AS DECIMAL (10,2))) AS DECIMAL (10,2))AS 'Average Score per Golf Course', User_Info.UserID, FirstName+' '+ LastName AS 'Full Name', Handicap`	By using the COUNT function, this command selects the number of times the player has inputted their scores. Afterwards, by using the CAST function, the average (AVG function) score will have 2 decimal points to have a more accurate average score per user. The first and last name will populate in the same field for easier viewing. Lastly, the query will show the current handicap of the user.

`FROM User_Info`
`JOIN Golf_Scores ON User_Info.UserID =`
`Golf_Scores.UserID`
`GROUP BY User_Info.UserID, FirstName,`
`LastName, Handicap`
`HAVING COUNT(Golf_Scores.UserID) > 1`
	This will join the UserID from the User_Info table with the UserID at the Golf_Scores table and this is necessary because the scores have to match with the user's Full Name or UserID. Then it will be using the GROUP BY function to group the UserID, First Name, Last Name, and Handicap. Lastly, by using the "HAVING" function for the accurate average, there needs to be more than 1 score for an accurate reading for the average score and will count the number of golf score submissions by using the COUNT function. If there was not a HAVING function, then "Jonathan Lee" would show up in the table having a scoring average of 90 shots because he only played once.

I generated this query in order to find the average scores per golf course. This is important because then the user can see which golf course was the hardest or easiest depending on average of golf scores and see if they want to play the golf course to challenge themselves or just have a good round of golf. The query below will demonstrate the number of rounds there have been played per golf course and their respective average scores.



```
SELECT COUNT(Golf_Scores.Scores) as '# of Scores per Course', CAST(AVG(CAST(Scores AS DECIMAL (10,2))) AS DECIMAL (10,2)) AS 'Average Score per Golf Course',
                Golf_Course.GolfCourseID, Golf_Course.GolfCourseName
        FROM Golf_Course
        JOIN Golf_Scores ON Golf_Course.GolfCourseID = Golf_Scores.GolfCourseID
        GROUP BY Golf_Course.GolfCourseID, GolfCourseName
```

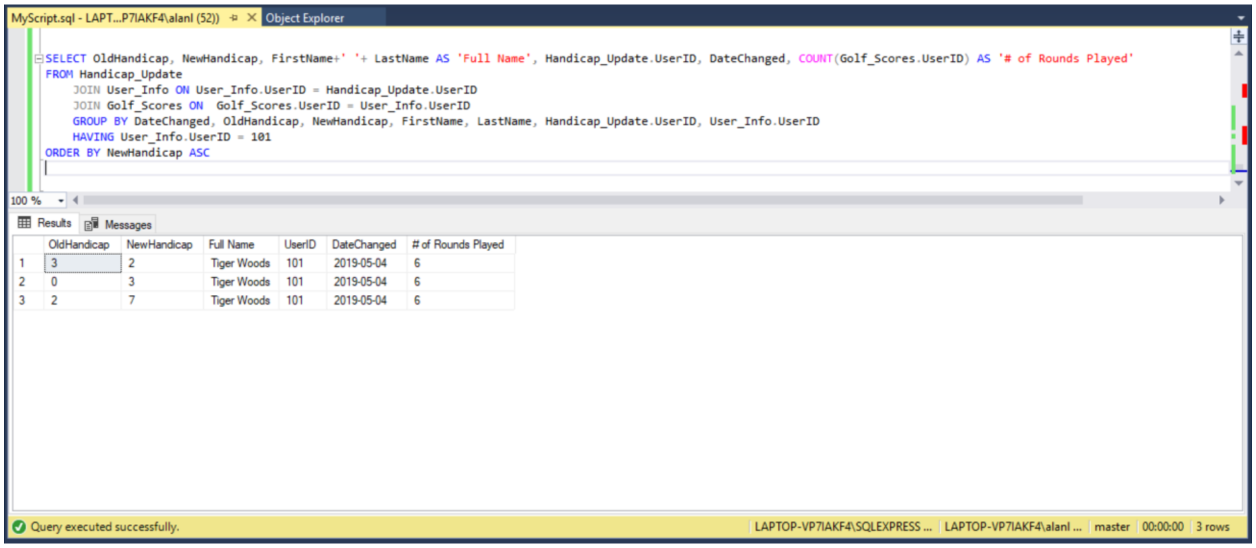
	# of Scores per Course	Average Score per Golf Course	GolfCourseID	GolfCourseName
1	7	71.43	401	Los Inkas Country Club
2	4	69.50	402	Avila Golf and Country Club
3	6	73.17	403	Augusta National
4	5	77.00	404	Country Club La Dehesa

```
Query executed successfully.    LAPTOP-VP7IAKF4\SQLEXPRESS ... | LAPTOP-VP7IAKF4\alanl ... | master | 00:00:00 | 4 rows
```

Code	**Description**
`SELECT COUNT(Golf_Scores.Scores) as '# of Scores per Course', CAST(AVG(CAST(Scores AS DECIMAL (10,2))) AS DECIMAL (10,2)) AS 'Average Score per Golf Course', Golf_Course.GolfCourseID, Golf_Course.GolfCourseName`	This command selects the number of times the player has inputted their scores, which will determine the number of golf rounds played per golf course by using the COUNT function in Scores from the Golf_Scores table. Afterwards, the average score will have 2 decimal points by using the CAST function and this will allow the database to show a more accurate average score per golf course. Afterwards, it will show the Course ID with their respective Golf Course Name.
`FROM Golf_Course JOIN Golf_Scores ON Golf_Course.GolfCourseID = Golf_Scores.GolfCourseID`	This will join the Golf Course ID from the Golf_Course table with the Golf Course ID at the Golf_Scores table and this is necessary because

`GROUP BY Golf_Course.GolfCourseID, GolfCourseName`	the scores have to match with the respective Golf Course ID. Then it will be using the GROUP BY function to group the Golf Course ID and Golf Course Name.

Lastly, the user will want to see their progress throughout their golf rounds. This can be shown by looking at their progress in their handicap level. They can see if it's increasing or decreasing. If it is decreasing then, then that means the user is getting better or vice versa. Since it is time consuming to hardcode the date per score, the trigger I created is an automated GETDATE function so when I updated the handicap to test out trigger it was all on the same date. For now, the handicap changes will have the same day in the query, but once there are more users updating their handicap on different days, then the users will be able to see their progression throughout their golf rounds. The query shown below will demonstrate the user of all of their handicap changes they have made in the database.



```
SELECT OldHandicap, NewHandicap, FirstName+' '+ LastName AS 'Full Name', Handicap_Update.UserID, DateChanged, COUNT(Golf_Scores.UserID) AS '# of Rounds Played'
FROM Handicap_Update
    JOIN User_Info ON User_Info.UserID = Handicap_Update.UserID
    JOIN Golf_Scores ON  Golf_Scores.UserID = User_Info.UserID
    GROUP BY DateChanged, OldHandicap, NewHandicap, FirstName, LastName, Handicap_Update.UserID, User_Info.UserID
    HAVING User_Info.UserID = 101
ORDER BY NewHandicap ASC
```

	OldHandicap	NewHandicap	Full Name	UserID	DateChanged	# of Rounds Played
1	3	2	Tiger Woods	101	2019-05-04	6
2	0	3	Tiger Woods	101	2019-05-04	6
3	2	7	Tiger Woods	101	2019-05-04	6

Query executed successfully. LAPTOP-VP7IAKF4\SQLEXPRESS ... LAPTOP-VP7IAKF4\alanl ... master 00:00:00 3 rows

Code	Description
`SELECT OldHandicap, NewHandicap, FirstName+' '+ LastName AS 'Full Name', Handicap_Update.UserID, DateChanged, COUNT(Golf_Scores.UserID) AS '# of Rounds Played'`	This will select the Old handicap, New Handicap from the Handicap_Update table and will add the First Name and Last Name columns into one field (Full Name). After, the query will show the User ID, and will show when the Date was changed. As mentioned before, since we don't want to hard code the numbers for date we use the GETDATE function in our trigger, so all the dates were the same for now. Then we counted the number of scores submitted into our database as "# of Rounds Played" by using the COUNT function.
`FROM Handicap_Update JOIN User_Info ON User_Info.UserID = Handicap_Update.UserID JOIN Golf_Scores ON Golf_Scores.UserID = User_Info.UserID GROUP BY DateChanged, OldHandicap, NewHandicap, FirstName, LastName, Handicap_Update.UserID, User_Info.UserID HAVING User_Info.UserID = 101 ORDER BY NewHandicap ASC`	This will join the User ID from the User_Info table to the User ID of Handicap_Table because we want to match the user's handicap with their names and scores. Later the second join will join the User ID from the User_Info table to the User ID of Golf_Scores because it is necessary that the user will be matched to the right golf scores they submitted. Later it will be grouped by date changed, old handicap, new handicap, First Name, Last Name, Handicap_Update.UserID, and User_Info.UserID.

	This will be all filtered buy just showing the User ID of 101 (Tiger Woods) and the New Handicap Update will be shown in ascending order by using the ASC function.

Summary and Reflection

I believe this app and website will be very useful to golfers who have the passion for golf and love to practice. They will be able to see how their numbers stack up with fellow golfers around the world. My biggest problem with this database strategy is that some of these statistics can be very complicated to write down or remember. For example, if a golfer gets really frustrated with their game, they might not want to input their scores because they might feel that it would worsen their stats. Also sometimes when the player gets frustrated, they tend to forget their stats of the specific fields causing them to guess and maybe having inaccurate results. Therefore, I have to make sure that this database will not be too specific, but detailed enough for golfers to use this database to help them improve on their game.

In the 5th project iteration, I had to add a new table (History_Update), and update my ERD, use cases and update my indexes. Overall, all the terms I have learned throughout the Project Iterations have helped me understand better my database and assisted me in simplifying my database. By making the database simpler to the golfer, the golfer will be more inclined in being more thorough with their golf statistics. Also by making the application cost money, it would give more incentive to the golfer to post their golf data onto the database. Having a free service in the application or the website would not motivate the user too much. By spending money on the application, they will upload their stats more often and will solve my problem of not getting enough data. Not only will it help their game, but this database will hold each player accountable of their progress in this wonderful sport.